

Anatomy of the Command Line

PLUG

2026Jan08 @ 19:00

PLUG

der.hans (https://floss.social/@FLOX_advocate)

Database Support Engineering Manager

Rackspace DBaaS, nee Object Rocket

<https://www.ObjectRocket.com/>

ordering

1. redirection
2. pre-command variable assignment
3. expansion*
4. pipes
5. commands

ordering w/ expansion

- redirection
- pre-command variable assignment
- expansion
 1. brace expansion - { }
 2. tilde expansion - ~
 3. parameter and variable expansion - \$VAR
 4. arithmetic expansion - \$(())
 5. command substitution - \$()
 6. word splitting - IFS
 7. pathname expansion - globs
- pipes
- commands

presentation conventions

```
### use a semicolon to end a command and start another on the same line  
;  
# dollar sign indicates a user shell  
$ I_am_me  
### hash indicates a root shell  
# I_have_the_power  
### triple hash is for comments
```

```
$ command1  
$ command2  
$ command3
```

```
$ command1; command2; command3
```

```
# sync; sync; sync; reboot
```

redirection

```
$ echo "UseRoaming no" >> /etc/ssh/ssh_config
$ echo "UseRoaming no" | sudo tee -a /etc/ssh/ssh_config
```

```
$ echo anke >file.txt; grep anke file.txt
anke
$ echo anke >file.txt; grep anke file.txt | grep anke >file.txt
$
$ echo anke >file.txt; ls -l file.txt; grep anke file.txt | grep anke > file.txt; ls -l file.txt
-rw-r--r-- 1 lufthans lufthans 5  4. Jan 22:41 file.txt
-rw-r--r-- 1 lufthans lufthans 0  4. Jan 22:41 file.txt
$
```

pre-command variable assignments

```
$ LANG=C
$ find /tmp
find: '/tmp': No such file or directory
$ LANG=de_DE.UTF-8 find /tmp | LANG=en_US sed -re 's/'
find: /tmp: Datei oder Verzeichnis nicht gefunden
sed: -e expression #1, char 2: unterminated `s' command
$
```

expansion

- brace expansion
- TPCAP (done in a left-to-right fashion, then quote removal)
 - tilde expansion
 - parameter and variable expansion
 - command substitution
 - arithmetic expansion
 - process substitution (where supported, e.g. *NIX boxen)
- word splitting
- pathname expansion

tokenizing

From the bash man page:

```
Only brace expansion, word splitting, and pathname expansion can in-  
crease the number of words of the expansion; other expansions expand a  
single word to a single word. The only exceptions to this are the ex-  
pansions of "$@" and "${name[@]}", and, in most cases, "$*" and  
"${name[*]}" as explained above (see PARAMETERS).
```

brace expansion

- bashism, not in sh
- left to right order is preserved

```
$ echo Number:{two,three,one}
Number:two Number:three Number:one
$ echo {,/usr}/{,s}bin
/bin /sbin /usr/bin /usr/sbin
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
$ ten=10; echo {1..$ten}
{1..10}
$ echo {1{0..9},20}
10 11 12 13 14 15 16 17 18 19 20
$ echo {a..e}
a b c d e
$ echo 1{0..9..2}
10 12 14 16 18
$
```

TPCAP

- expanded in the order encountered, left to right
 - tilde expansion
 - parameter and variable expansion
 - command expansion
 - arithmetic expansion
 - process substitution

tilde expansion

```
$ ls -ld ~/../../var/{lib,log,cache}
drwxr-xr-x 23 root root 4096 21. Jul 21:36 /home/plug/../../var/cache
drwxr-xr-x 71 root root 4096  9. Dez 02:16 /home/plug/../../var/lib
drwxr-xr-x 15 root root 4096  2. Jan 08:00 /home/plug/../../var/log
$ echo ~$USER
plug
$
```

- beware use in scripts

parameter expansion

- aka variables

```
$ echo $HOME
/home/plug
$ echo $horse_with_no_name
$
```

- lots of fun string manipulation that can be done during parameter expansion
 - search the bash manpage for :-

```
$ echo ${NOTHING:-something}
something
$ NOTHING=something
$ echo ${NOTHING:+lurch}
lurch
$ echo ${NOTHING^^}
SOMETHING
```

arithmetic expansion

- dollar sign, two opening parens, **integer expression**, two closing parens: `$((math))`
- normal math ordering
- the expression is implicitly double quoted

```
$ echo $(( 3 + 1 ))  
4
```

- integer only

```
$ echo $(( 3 + 1.5 ))  
bash: 3 + 1.5 : syntax error: invalid arithmetic operator (error token is ".5 ")  
4  
$ echo $(( 4 / 3 ))  
1  
$
```

math nesting

- nesting works, but use parenthesis for ordering

```
$ echo $(( $( ( 3 + 5 ) ) * 2 ))  
16  
$ echo $(( ( 3 + 5 ) * 2 ))  
16
```

- a double quote inside expression remains a double quote
- parameter and variable expansion, command substitution, and quote removal happen
- can leave sigil off variables

```
$ rush=2112; echo $(( 2 * $rush ))  
4224  
$ cent=19; year=84; orwell=$(( cent * 100 + year )); echo $orwell  
1984  
$
```

subshell

- opening paren, command list, closing paren: **(cmd1; cmd2)**
- runs inline
- creates a new env that doesn't change parent environment

```
$ somevar=someval; ( somevar=otherval ); echo $somevar
someval
$
```

- see [Compound Commands in the bash manpage](#)

command substitution

- dollar sign, opening paren, command list, closing paren: `$(cmd1; cmd2)`
- replaces the command with the results of the command
- creates a new env that doesn't change parent environment
- NOTE: do not use backticks
 - nesting with backticks is a quoting nightmare
- quotes are protected when using dollar paren

```
$ echo >file.$( date +%Y%b%d ).adoc
$ ls -l file.$( date +%Y%b%d ).adoc
-rw-r--r-- 1 lufthans lufthans 1 Sep 14 18:09 file.2023Sep14.adoc
$
```

```
cmdOut=$( command $( getArgs ) )
grepOut=$( grep -i PLUG $( find $HOME/ | grep -i plug | grep .adoc ) )
```

detours

- nesting: command substitution in arithmetic

```
$ echo $(( $( grep -c . /etc/motd ) * 2 ))  
10  
$
```

- nesting: command substitution with redirect shenanigans

```
$ echo -n $( echo foo >>file.txt ) >file.txt  
$ ls -l file.txt  
-rw-r--r-- 1 lufthans lufthans 0  9. Aug 23:10 file.txt  
$ echo $( echo foo >>file.txt; ls -l file.txt ) >file.txt  
$ cat file.txt  
-rw-r--r-- 1 lufthans lufthans 4 9. Aug 23:11 file.txt  
$
```

process substitution

- system must support named pipes (FIFOs) or /dev/fd
- use for commands that really, really want a filename rather than STDIN

```
$ diff <(ls /etc/ | tail -3) <(ls /etc/ | tail -4)
0a1
> NetworkManager
$
```

word splitting

- splits on text not in double quotes
- defaults to space, tab and newline
- IFS - Internal Field Separator
- scans results of parameter expansion, command substitution, and arithmetic expansion that wasn't quoted

```
$ ls /etc/{issue,motd}
/etc/issue /etc/motd
$ echo ls /etc/{issue,motd}
ls /etc/issue /etc/motd
$
```

```
ls $( echo /etc/resolv.conf /etc/issue )
```

pathname expansion

- globbing

```
$ ls -d {/etc/w*,/usr/s*}
/etc/wgetrc      /etc/wpa_supplicant /usr/share
/etc/wodim.conf /usr/sbin           /usr/src
$ ls -d {/etc/w*,/usr/s*} | wc -w
6
$
```

pipes

- don't use pipes for parallelization, they're for redirecting STDOUT to STDIN

```
$ rm anke.txt; ls -l anke.txt >&2 | cat anke.txt >&2 | echo anke >anke.txt
anke
-rw-r--r-- 1 lufthans lufthans 5  6. Jan 23:33 anke.txt
$
```

```
$ ( LANG=en_US; find /tmp 2>&1 | sed -re 's@tmp@tmp@' 1>&2 | ls -l /etc/issue | date )
Thu Sep 14 17:30:56 MST 2023
find: '/tmp': No such file or directory
$
```

- see `$(PIPESTATUS)`

command

- and finally we run the command or equivalent

functions, aliases

- e.g. commands

extra credit

grouping with curly brackets

- see { list; } under Compound Commands in the bash manpage
- runs in current environment, but groups the commands

```
$ somevar=someval; { somevar=otherval; ls -l /dev/null; }; echo $somevar
crw-rw-rw- 1 root root 1, 3 Aug 31 12:34 /dev/null
otherval
$
```

regex

- passed from shell to comand, but beware interpretation as globs

```
$ touch f.{1,2,3}.txt
$ ls
f.1.txt f.2.txt f.3.txt
$ echo fff >file.txt
$ grep f.* file.txt
$ echo f.* file.txt
f.1.txt f.2.txt f.3.txt file.txt
$ grep f file.txt
fff
$
```

single quote before a command

```
$ \ls
```

```
$ ls
file.txt f.3.txt f.2.txt f.1.txt
$ \ls
f.1.txt f.2.txt f.3.txt file.txt
$ type ls
ls is aliased to `ls -r'
$
```

space at beginning

```
$ ls
```

```
$ echo
```

```
$ ls
```

```
file.txt f.3.txt f.2.txt f.1.txt
```

```
$ echo 2
```

```
2
```

```
$ history | tail -3
```

```
2013 echo
```

```
2014 echo 2
```

```
2015 history | tail -4
```

```
$
```

QUIZ TIME !!!

examples

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- `ls f*`
- `grep foo $(ls f* | grep -v w)`
- `LANG=en_US ls /t{tmp/,etc/,bin/*}`
- `ls -d ~/IMG*$((2112 + 1984))*{jpg,JPG,jpeg,JPEG}`

examples zwei

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- `ls -l file.out >file.out`
- `ls -l file.out | tee file.out`
- `sudo ls -l file.out >file.out`

examples trois

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- (for i in \$(find /etc -type f 2>/dev/null | grep sources.list); do tstamp=\$(date -d@\$(stat -c %Y %i) +%Y%b%d); echo cp -pi \$i /tmp/\$(echo "\${i}" | tr / _).\${tstamp}; done)

examples quattro

- * redirection
- * pre-command variable assignment
- * expansion
 - ** brace / tilde / parameter / arithmetic / cmd sub / process sub
 - ** word splitting - IFS
 - ** pathname expansion - globs
- * pipes
- * commands

- `echo $((2112 * 2)) $HOME | sed -re 's//z/g'`

Resources

- bash man page
- Advanced Bash-Scripting Guide